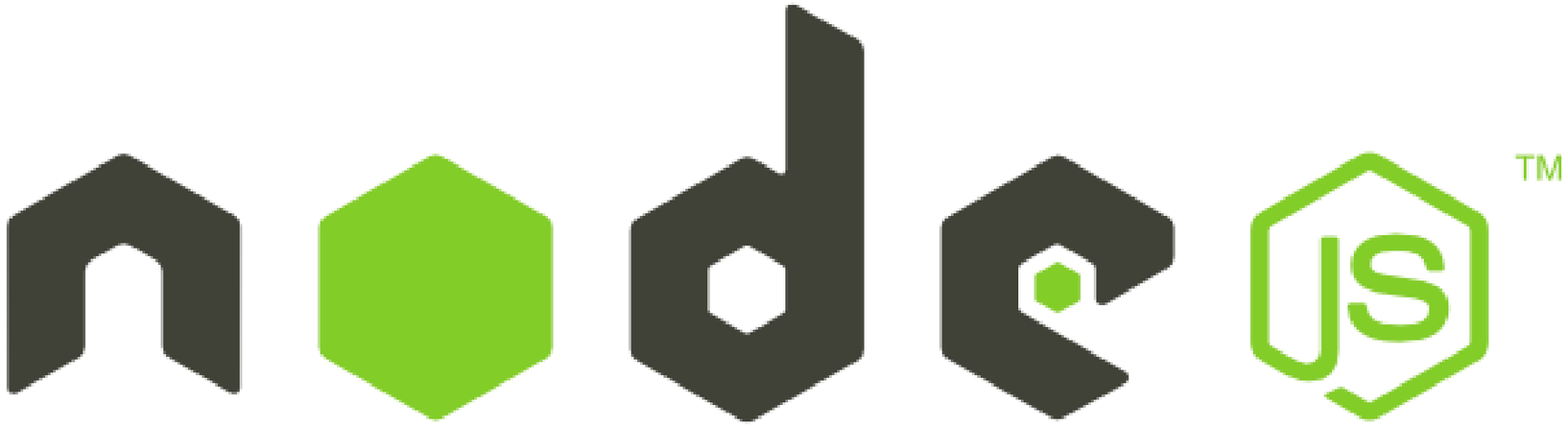


Introduction to Node.js

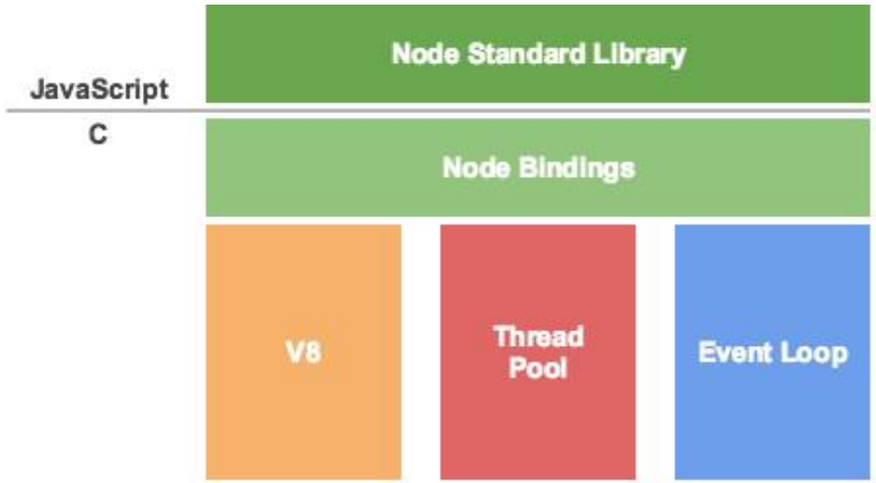
Frank Walsh
Diarmuid O'Connor

Agenda

- What is node.js
- Non Blocking and Blocking
- Event-based processes
- Callbacks in node
- Node Package Manager(NPM)
- Creating a node app
- Introduction to Express

What's Node: Basics

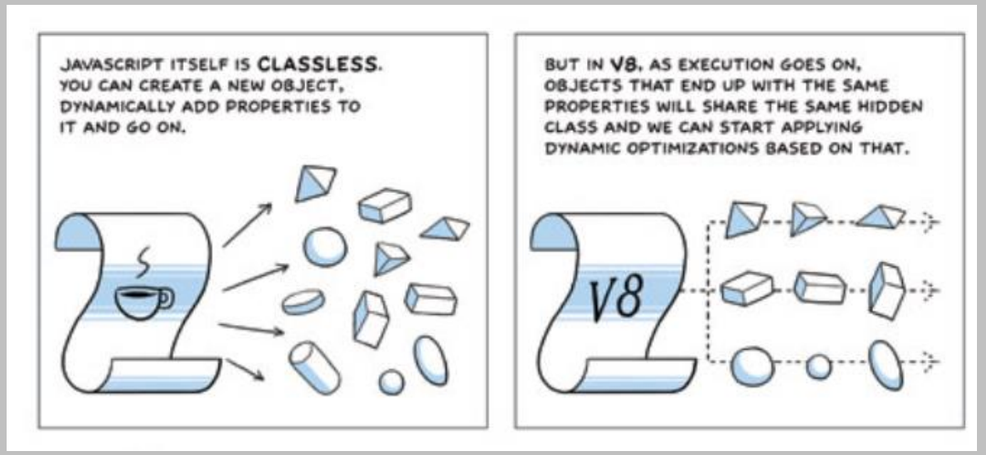
- A Javascript runtime. “Server side JS”
- The “.js” doesn’t mean that it’s written completely in JavaScript.
  - approx. 40% JS and 60% C++
- Ecosystem of packages (NPM)
- Official site: “Node's goal is to provide an easy way to build scalable network programs”.
- Single Threaded, Event based
  - Supports concurrency using events and callbacks…



What’s Node: V8.

- Embedded C++ component
- Javascript virtual machine.
- Very fast and platform independent
- Find out a bit about it's history here:

http://www.google.com/googlebooks/chrome/big_12.html



What is Node.js: Event-based

- Input/Output (io) is slow.
  - Reading/writing to data store, network access.
  - Read 4K randomly from SSD* 150,000 ns ~1GB/sec SSD
  - Round trip over network within same datacenter 500,000 ns
  - Send packet US->Netherlands->US 150,000,000 ns
- CPU operations are fast.
  - L1 cache reference 0.5 ns
  - L2 cache reference 7 ns

- I/O operations detrimental to highly concurrent apps (e.g. web applications)
- Solutions to deal with this are:
  - **Blocking code** combined with multiple threads of execution (e.g. Apache, IIS)
  - **Non-blocking**, **event-based code** in single thread (e.g. NGINX, Node.js)

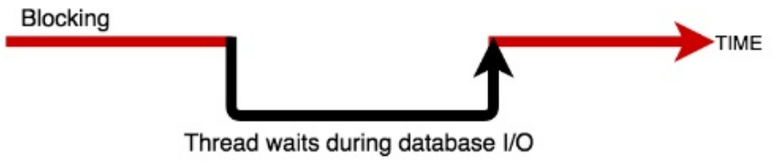
Source: https://gist.github.com/jboner/2841832

$1ns = 10^{-9}\ s\ (0.000000001s)$

Blocking/Non-blocking Example

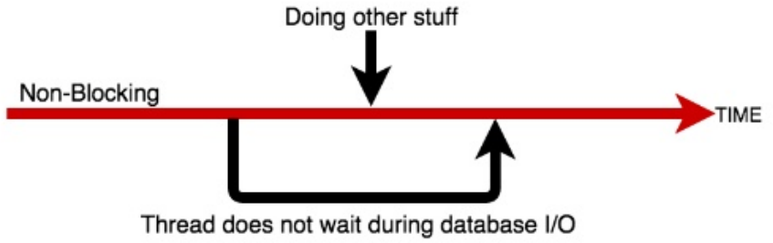
Blocking

1. Read from file and set equal to contents
2. Print Contents
3. Do other stuff...



Non-blocking

1) Read from File
   Whenever read is complete, print contents
2) Do other stuff...



Blocking/Non-blocking: JS

Blocking

```
import fs from 'fs';

const contents = fs.readFileSync('./readme.md', 'utf8');
console.log(contents);
console.log('Doing something else');
```

Console output

Hello World……
Doing something else

Non-blocking

callback

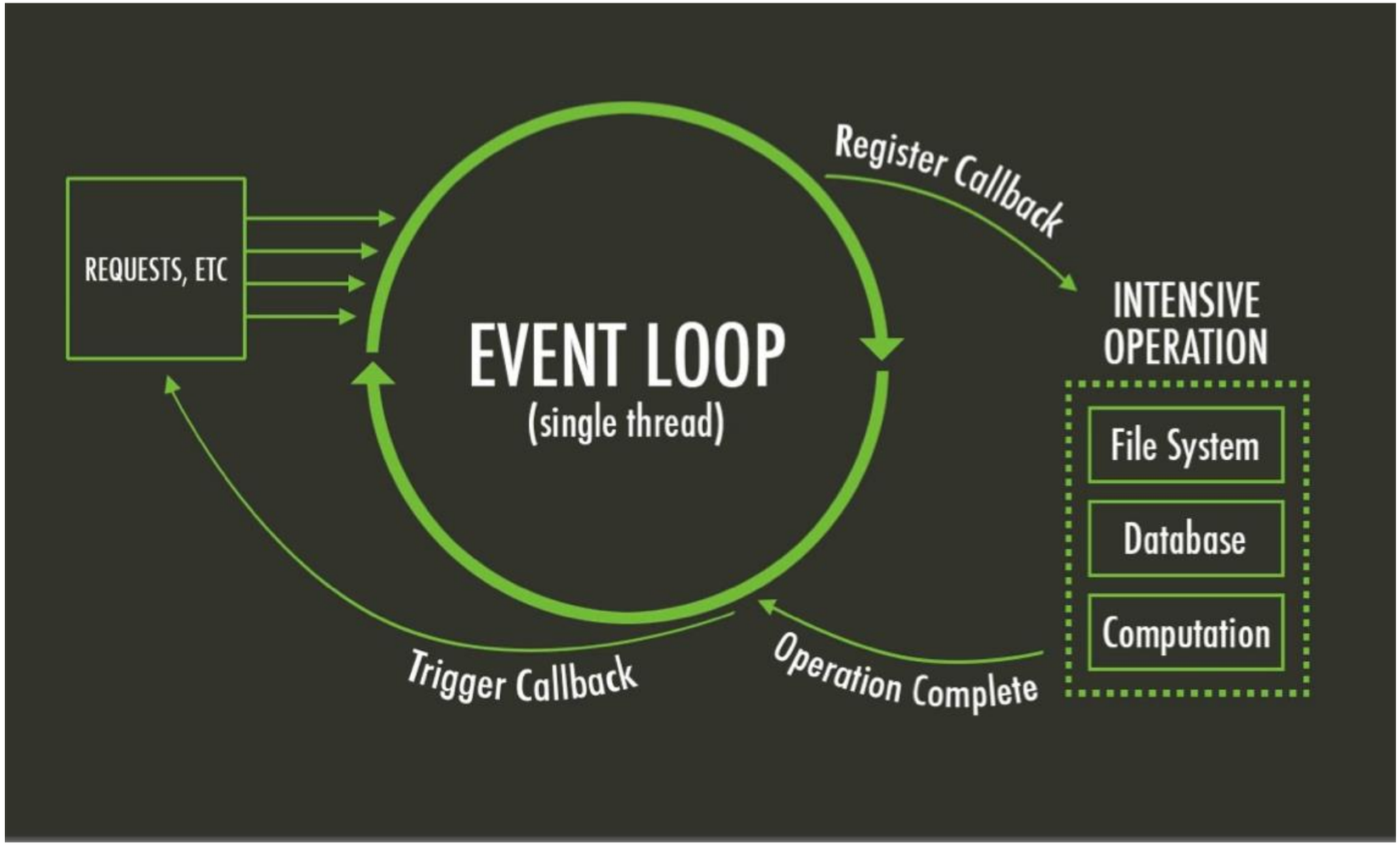
```
import fs from 'fs';
fs.readFile('./text.txt','uft8', (err, contents) => {
    console.log(contents);
});
console.log('Doing something else');
```

Console output

Doing something else
Hello World ……

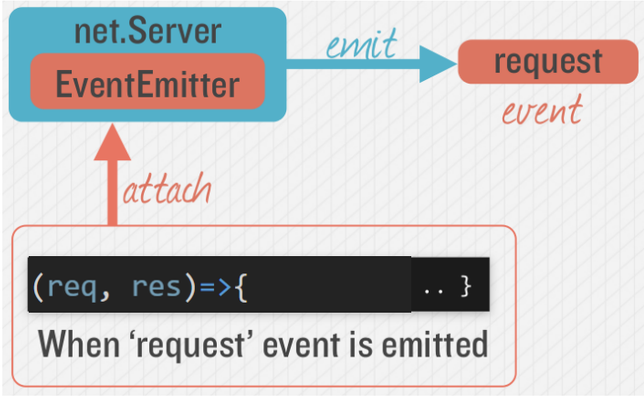
The Node Event Loop and Callbacks

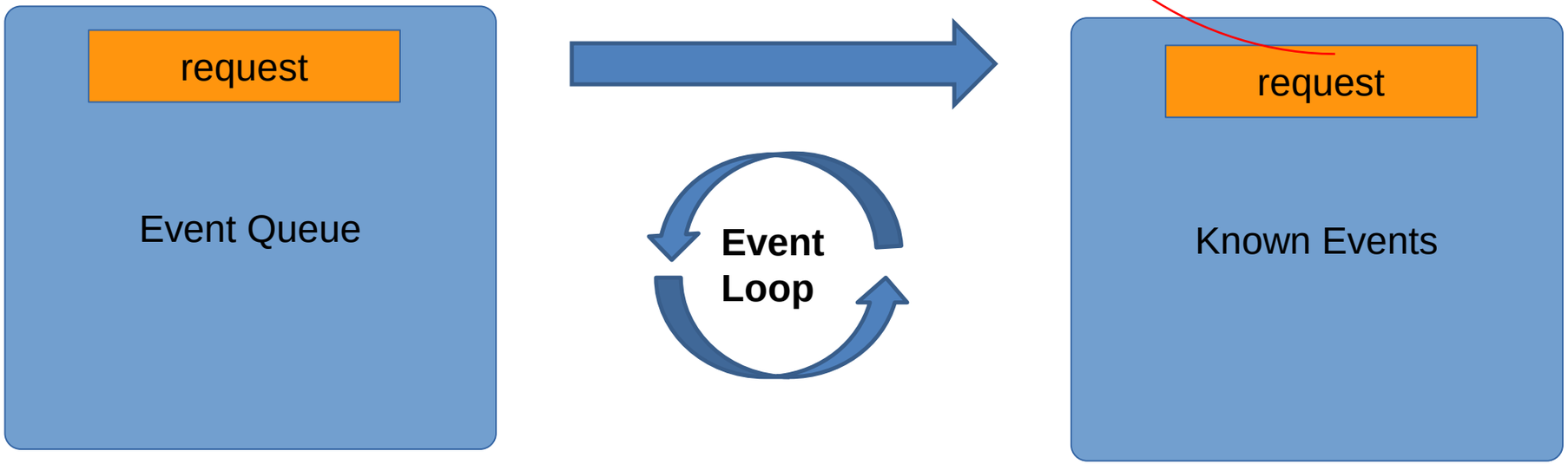
- A **Callback** is a function called at the completion of a given task. This prevents any blocking, and allows other code to be run in the meantime
- The Event Loop checks for known events, registers Callbacks and, triggers callback on completion of operation



Node.js - Simple HTTP Server

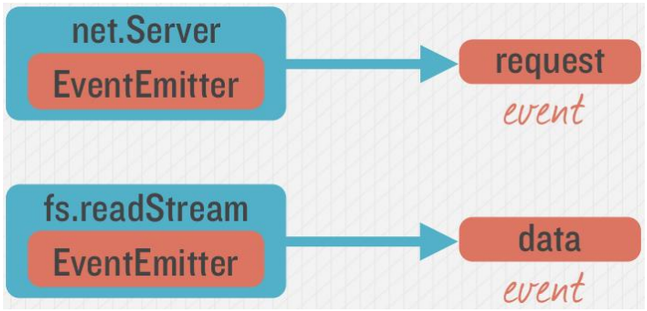
```
1   import http from 'http';
2   const port=8080;
3
4   var server = http.createServer((req, res)=>{
5       response.writeHead(200);
6       response.end("Hello World!");
7   });
8
9   server.listen(port);
10  console.log(`Server running at ${port}`);
```





Emitting Event in Node

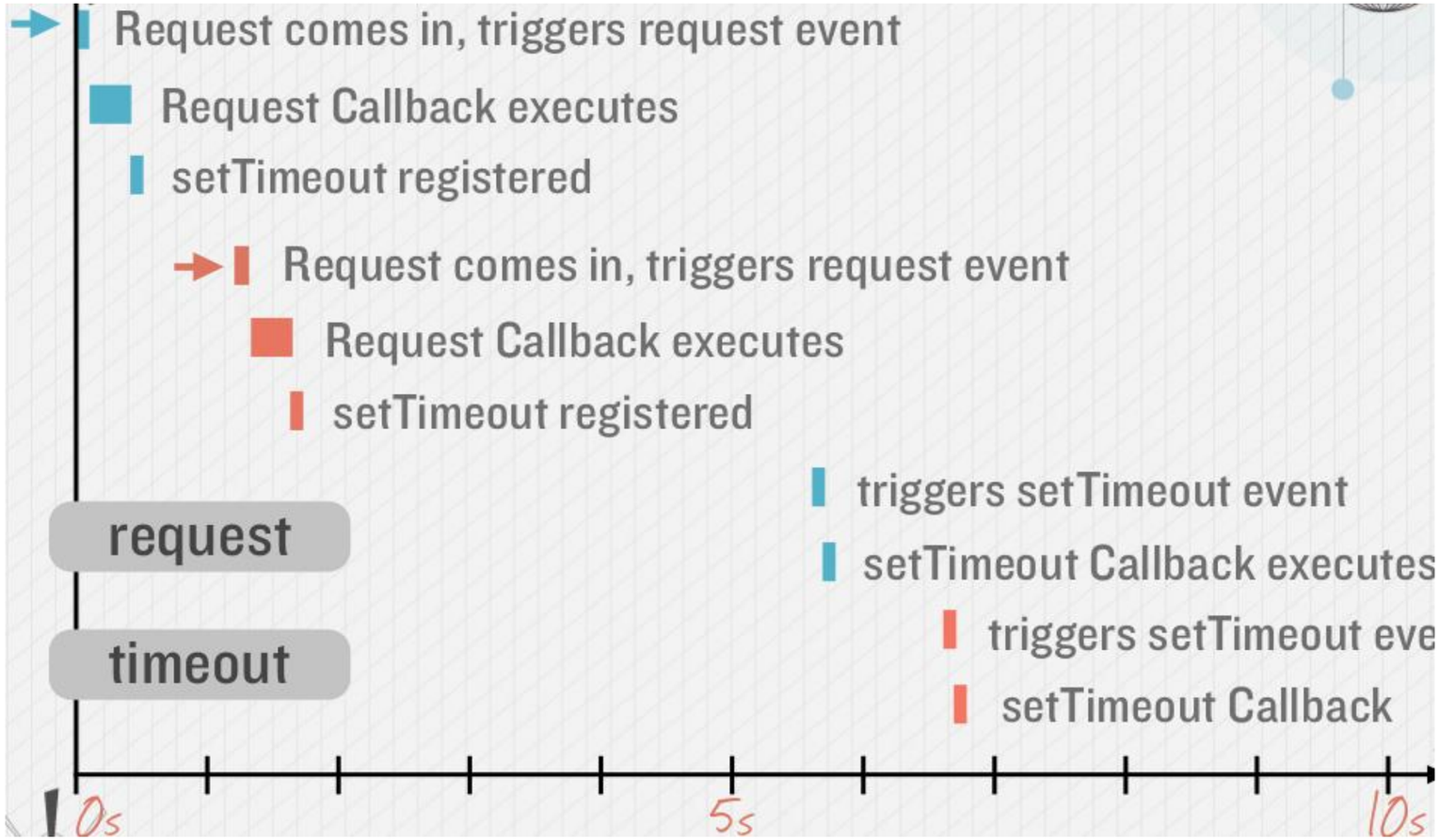
Many objects can emit events in node.



Example – Hello/Goodbye Callback

Callback Timeline, Non Blocking

Timing example: 2 requests to web application (indicated by red and blue in diagram)



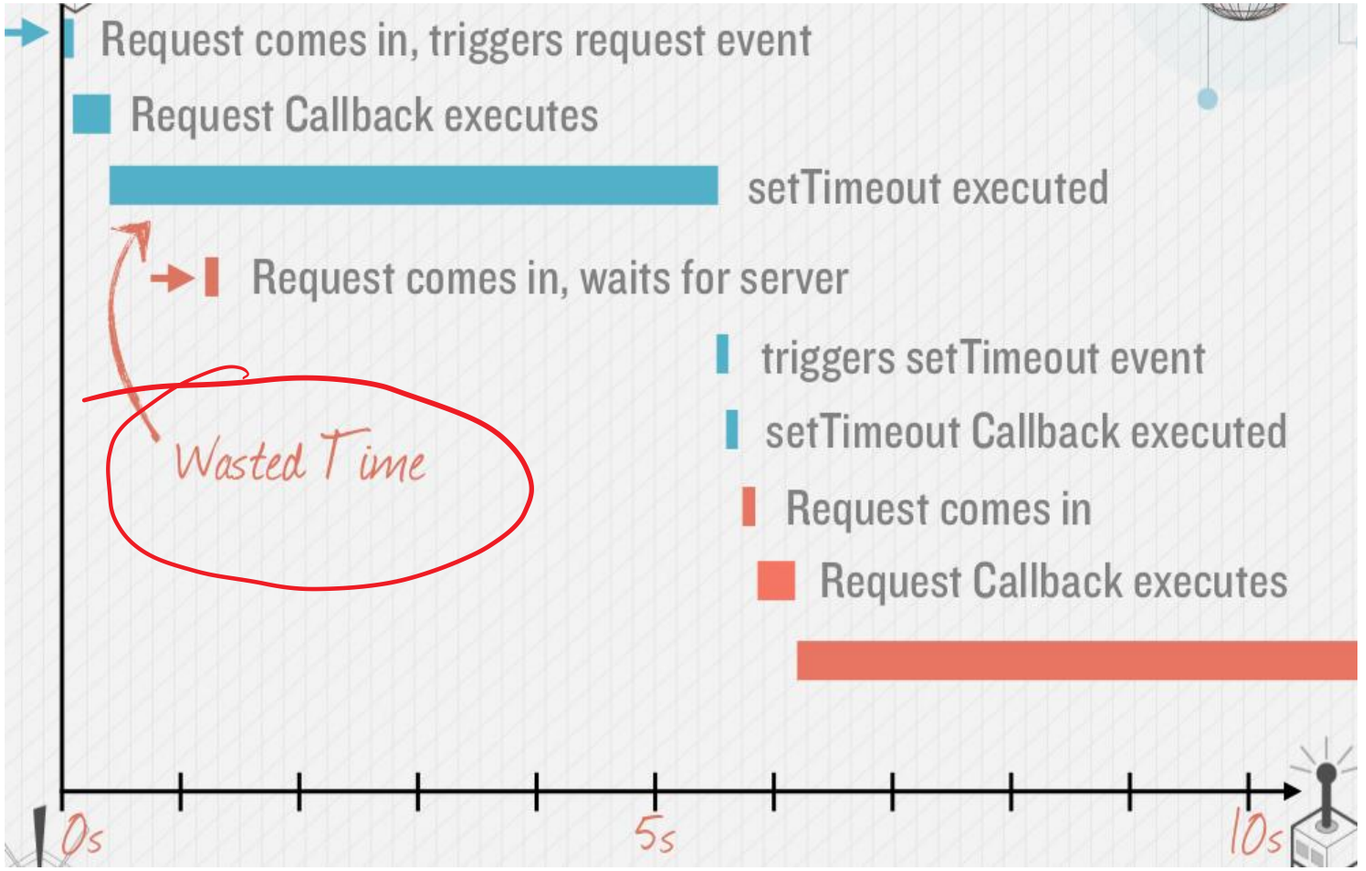
Avoid Blocking Calls in Node.js apps

- setTimeout in previous slide is an example of an asynchronous, non-blocking call.
- Avoid potential blocking/ synchronous calls
- **Activity likely to be blocking should be called asynchronously.**

Examples:

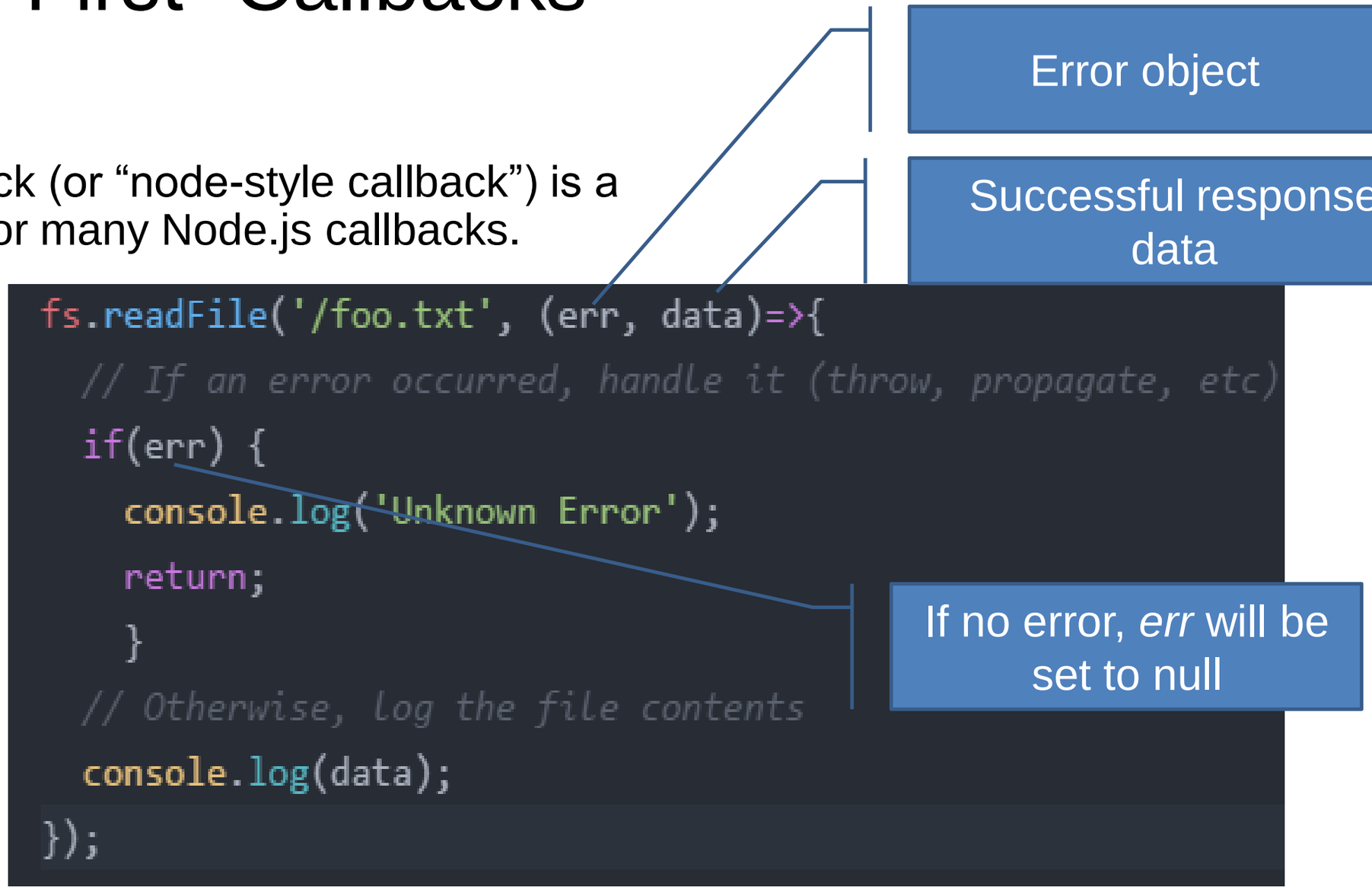
- Calls to 3rd party Web Services
- Database queries
- Computationally expensive operations (image file processing)

What if setTimeout() blocked…



Node “Error First” Callbacks

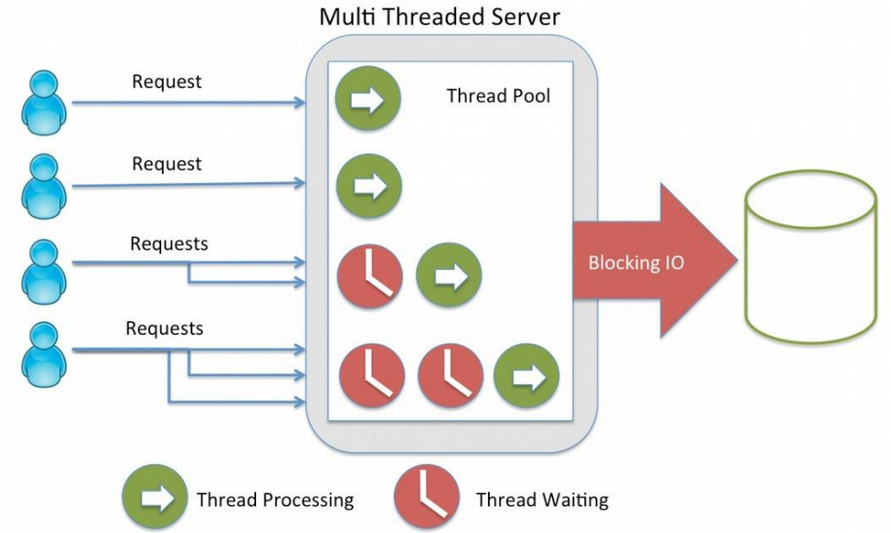
The “error-first” callback (or “node-style callback”) is a standard convention for many Node.js callbacks.



Blocking vs. Non-blocking: Web Servers

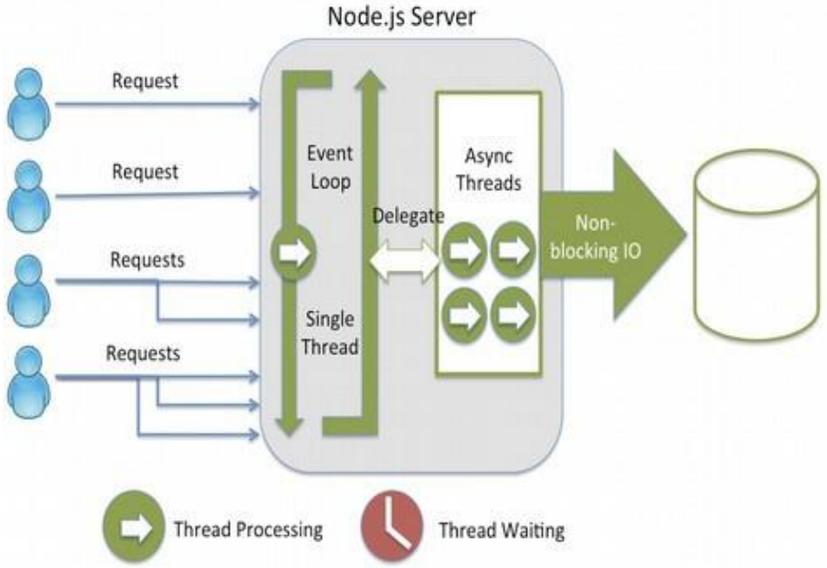
Threads consume resources

- Memory on stack
- Processing time for context switching etc.



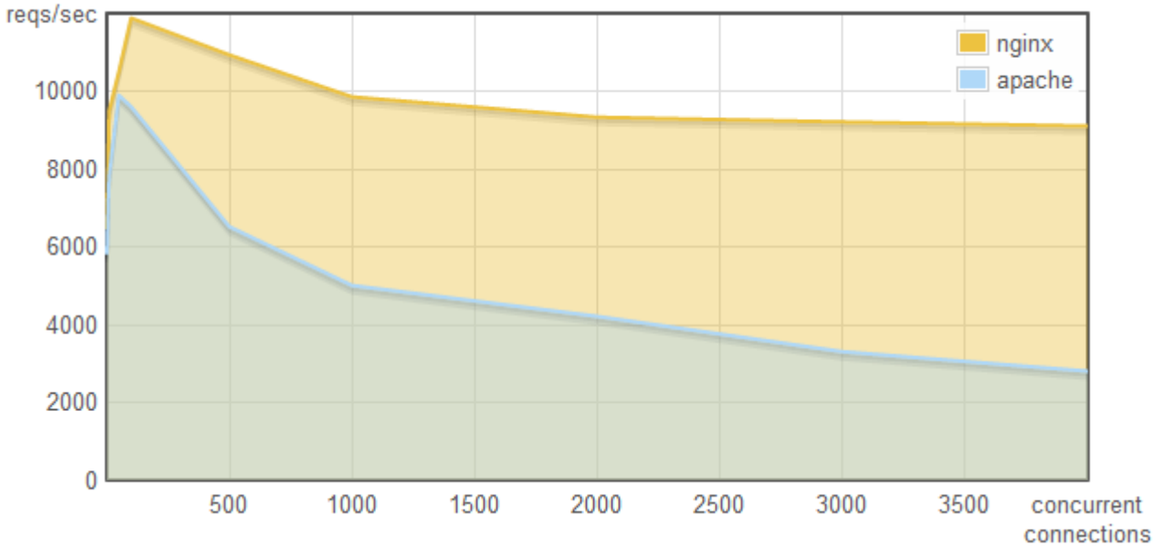
No thread management on single threaded apps

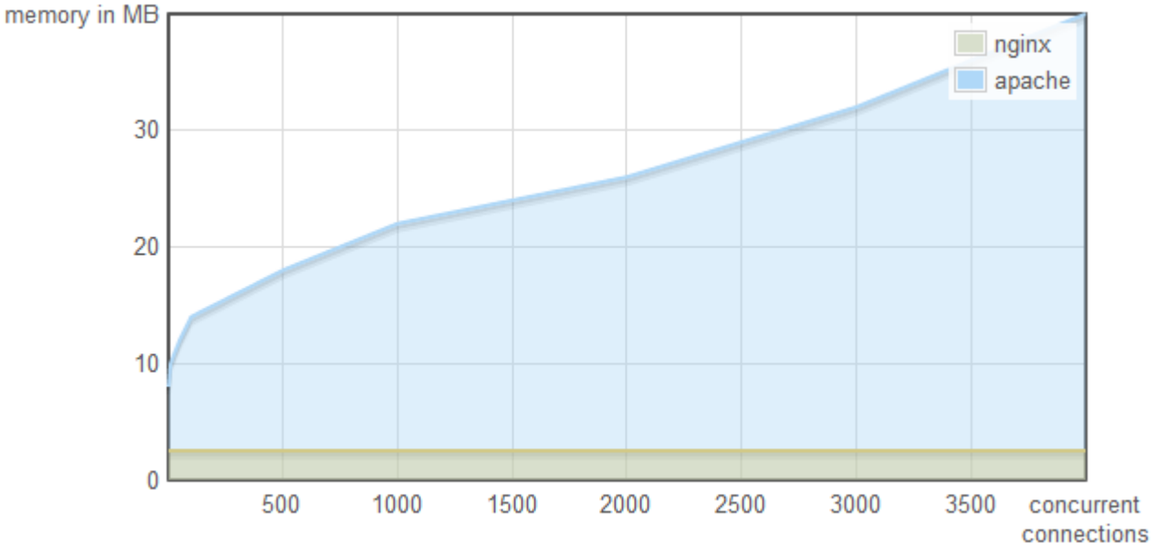
- Just execute “callbacks” when event occurs



Why does it matter...

- This is why:





http://blog.webfaction.com/a-little-holiday-present

Node Modules

New Programs Makers

npm Enterprise Products Solutions Resources Docs Support

npm Search packages Search Join Log In

Build amazing things

Essential JavaScript development tools that help you go to market faster and build powerful applications using modern open source code.

See plans Join for free

Node Modules

- Node has a small core API
- Most applications depend on third party modules
- Curated in online registry called the Node Package Manager system (NPM)
- NPM downloads and installs modules, placing them into a **node_modules** folder in your current folder.

NPM init

- You can use NPM to manage your node projects
- Run the following in the root folder of your app/project:

  npm init
- This will ask you a bunch of questions, and then create a package.json for you.
- It attempts to make reasonable guesses about what you want things to be set to, and then writes a package.json file with the options you've selected.

Node Modules

- To install NPM modules, navigate to the application folder and run “npm install”. For example :

  npm install express --save
- This installs into a “**node_module**” folder in the current folder.
- The **--save** bit updates your package.json with the dependency
- To use the module in your code, use:

  ```
  import express from 'express';
  ```
- This loads express from local **node_modules** folder.

Global Node Modules

- Sometimes you may want to access modules from the shell/command line.
- You can install modules that will execute globally by including the **'-g'.**
- Example, **Grunt** is a Node-based software management/build tool for Javascript.

  npm install -g grunt-cli
- This puts the **“grunt”** command in the system path, allowing it to be run from any directory.

NPM Common Commands

Common npm commands:

– **npm init** *initialize a package.json file*

– **npm install <package name> -g** *install a package, if –g option is given package will be installed globally,* ***--save*** *and* ***--save-dev*** *will add package to your dependencies*

– **npm install** *install packages listed in package.json*

– **npm ls –g** *listed local packages (without –g) or global packages (with –g)*

– **npm update <package name>** *update a package*

Creating your own Node Modules

- We want to create the following module called **greeting.js:**

```
const hello = function() {
console.log("hello!");
}
export default hello;
```

Export defines what import returns

- To access in our application, **index.js:**

```
import hello from './custom_hello';
hello();
```

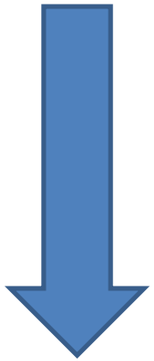
Creating your own Node Modules

- Exporting Multiple Properties

Config.js

```
const env = process.env;

export const nodeEnv = env.NODE_ENV || 'development';

export const logStars = function(message) {
  console.info('**********');
  console.info(message);
  console.info('**********');
};

export default {
  port: env.PORT || 8080,
  host: env.HOST || '0.0.0.0',
  get serverUrl() {
    return `http://${this.host}:${this.port}`;
  }
};
```

- Accessing in other scripts



```
import config from './config';
import { logStars, nodeEnv } from './config';


logStars(`Port is ${config.port},  host is ${config.host}, environment is ${nodeEnv}`);
console.info(`Contact api available at ${config.serverUrl}/api/contests`)
```

The import search

- Import searches for modules based on path specified:

```
import myMod from ('./myModule');  //current dir
import myMod from ('../myModule'); //parent dir
import myMod from ('../modules/myModule');
```

- Just providing the module name will search in **node_modules** folder

```
import myMod from ('myModule')
```

The Express Package

express

4.16.4 • Public • Published 5 months ago

Readme | 30 Dependencies | 31,220 Dependents | 261 Versions

Fast, unopinionated, minimalist web framework for node.

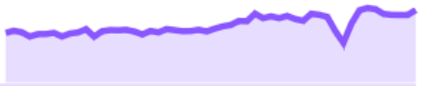

```
var express = require('express')
var app = express()

app.get('/', function (req, res) {
  res.send('Hello World')
})

app.listen(3000)
```

install

```
> npm i express
```

weekly downloads

7,597,647



version	license
4.16.4	MIT

open issues	pull requests
115	59

homepage	repository
expressjs.com	github

last publish

4 months ago

What Express Gives Us...

- Parses arguments and headers
- Easy Routing
  - Route a URL to a callback function
- Sessions
- File Uploads
- Middleware…

Simple Express App (index.js)

```
import express from 'express';

const app = express();

app.use(express.static('public'));

app.listen(8080, () => {
  console.info('Express listening on port', 8080);
});
```

Loads Express module

Instantiates Express server

Define static content for HTTP GET

Getting Started with Express

- Installing Express

```
[local install] C:\> npm install express --save
[global install] C:\> npm install express -g
```

Express Configuration

Express allows you to easily configure your web app behaviour...

```
// allow serving of static files from the public directory
app.use(express.static('/public'));
// configure to parse application/json
app.use(bodyParser.json());
// configure to parse application/x-www-form-urlencoded
app.use(bodyParser.urlencoded({ extended: true }));
```

(/*)?

Routing Examples

Syntax follows the pattern:

App.[verb](path, (req,res)=>{});

```
import express from 'express';

const app = express();

app.use(express.static('public'));

app.get('/contacts', (req,resp)=>{resp.end('I should really be a collection of contacts');});

app.listen(8080, () => {
  console.info('Express listening on port', 8080);
});
```

```
// Other Route examples
app.post('/contacts', createContact);
app.get('/app/:app', routes.getapp);

//Catch-all
app.all('/private(/*)?', requiresLogin);
```

HTTP POST request

Parametised URL. Accepts :app route argument

Catch-all – works for all HTTP verbs

Node Applications Structure

Structuring Node Apps

- Node Server Code needs to be structured
  - Manage code base
  - Keeps code maintainable
  - Nodes packaging system supports this approach
- Typical Node.js application code:
  - main app code
  - api implementation code
  - helper code

Example Approach:

- Use a “project root” folder is the top level and contains the “entry point” or main server code
  - Always run npm in this folder to ensure just one node_modules folder
  - Use a **public** folder within the node folder for any static content

Basic Node App Structure